SPECIFICATION

System and Method for Recognizing Touch Typing under Limited Tactile Feedback Conditions

Cross Reference to Related Applications

Referenced-applications

09/236,513 1999-01-25 US 5,463,388 1993-01-29 US 5,812,698 1997-07-14

US 5,818,437 1995-07-26 US 6,137,908 1994-06-29 US 6,107,997 1996-06-27

US

Background of the Invention

1. Field of the Invention

[0001]   The present invention pertains to typing recognition systems and methods, and more particularly to recognition of typing in air or on a relatively smooth surface that provides less tactile feedback than conventional mechanical keyboards.

2. The Related Art

[0002]

Typists generally employ various combinations of two typing techniques: *hunt and peck* and *touch typing* . When hunting and pecking, the typist visually searches for the key center and strikes the key with the index or middle finger. When touch typing, the fingers initially rest on home row keys, each finger is responsible for striking a certain column of keys and the typist is discouraged from looking down at the keys. The contours and depression of mechanical keys provide strong tactile feedback that helps typists keep their fingers aligned with the key layout. The finger motions of touch typists are ballistic rather than guided by a slow visual

search, making touch typing faster than hunt and peck. However, even skilled touch typists occasionally fall back on hunt and peck to find rarely-used punctuation or command keys at the periphery of the key layout.

[0003]   Many touchscreen devices display *pop-up* or *soft* keyboards meant to be activated by lightly tapping a displayed button or key symbol with a finger or stylus. Touch typing is considered impractical on such devices for several reasons: a shrunken key layout may have a key spacing too small for each finger to be aligned with its own key column, the smooth screen surface provides no tactile feedback of finger/key alignment as keys are struck, and most touchscreens cannot accurately report finger positions when touched by more than one finger at a time. Such temporal touch overlap often occurs when typing a quick burst of keys with both hands, holding the finger on modifier keys while striking normal keys, or attempting to rest the hands. Thus users of touchscreen key layouts have had to fall back on a slow, visual search for one key at a time.

[0004]   Since touchscreen and touch keyboard users are expected to visually aim for the center of each key, typing recognition software for touch surfaces can use one of two simple, nearly equivalent methods to decide which key is being touched. Like the present invention, these methods apply to devices that report touch coordinates interpolated over a fine grid of sensors rather than devices that place a single large sensor under the center of each key. In the first method, described in U.S. Pat. Application Serial No. 09/236,513 by Westerman and Elias, the system computes for each key the distance from key center to the sensed touch location. The software then selects the key nearest the finger touch. In the second method, described in U.S. Pat. No. 5,463,388 to Boie *et al.* , the software establishes a rectangle or bounding box around each key and decides which, if any, bounding box the reported touch coordinates lie within. The former method requires less computation, and the latter method allows simpler control over individual key shape and guard bands between keys, but both methods essentially report the key nearest to the finger touch, independent of past touches. Hence we refer to them as 'nearest key' recognizers.

[0005]   Unlike touchscreens, the multi-touch surface (MTS) described by Westerman and Elias in Serial No. 09/236,513 can handle resting hands and temporal finger overlap during quick typing bursts. Since the MTS sensing technology is fully scalable, an MTS can easily be built large enough for a full-size QWERTY key layout. The only remaining barrier to fast touch typing on an MTS is the lack of tactile feedback. While it is possible to add either textures or compressibility to an MTS to enhance tactile feedback, there are two good reasons to keep the surface firm and smooth. First, any textures added to the surface to indicate key centers can potentially interfere with smooth sliding across the surface during multi-finger pointing and dragging operations. Second, the MTS proximity sensors actually allow zero-force typing by sensing the presence of a fingertip on the surface whether or not the finger applies noticeable downward pressure to the surface. Zero-force typing reduces the strain on finger muscles and tendons as each key is touched.

[0006]   Without rich tactile feedback, the hands and individual fingers of an MTS touch typist tend to drift out of perfect alignment with the keys. Typists can limit the hand drift by anchoring their palms in home position on the surface, but many keystrokes will still be slightly off center due to drift and reach errors by individual fingers. Such hand drift and erroneous finger placements wreak havoc with the simple 'nearest key' recognizers disclosed in the related touchscreen and touch keyboard art. For example, if the hand alignment with respect to the key layout drifts by half a key-spacing (~9mm or 3/8"), all keystrokes may land halfway between adjacent keys. A 'nearest key' recognizer is left to choose one of the two adjacent keys essentially at random, recognizing only 50% of the keystrokes correctly. A spelling model integrated into the recognizer can help assuming the typist intended to enter a dictionary word, but then actually hinders entry of other strings. Thus there exists a need in the touchscreen and touch keyboard art for typing recognition methods that are less sensitive to the hand drift and finger placement errors that occur without strong tactile feedback from key centers.

[0007]   For many years, speech, handwriting, and optical character recognition systems have employed spelling or language models to help guess users' intended words

when speech, handwriting, or other input is ambiguous. For example, in U.S. Pat. No. 5,812,698 Platt *et al.* teach a handwriting recognizer that analyzes pen strokes to create a list of probable character strings and then invokes a Markov language model and spelling dictionary to pick the most common English word from that list of potential strings. However, such systems have a major weakness. They assume all user input will be a word contained in their spelling or language model, actually impeding entry of words not anticipated by the model. Even if the user intentionally and unambiguously enters a random character string or foreign word not found in the system vocabulary, the system tries to interpret that input as one of its vocabulary words. The typical solution is to provide the user an alternative (often comparatively clumsy) process with which to enter or select strings outside the system vocabulary. For example, U.S. Pat. No. 5,818,437 to Grover *et al.* teaches use of a dictionary and vocabulary models to disambiguate text entered on a 'reduced' keyboard such as a telephone keypad that assigns multiple characters to each physical key. In cases that the most common dictionary word matching an input key sequence is not the desired word, users must select from a list of alternate strings. Likewise, users of speech recognition system typically fall back on a keyboard to enter words missing from the system's vocabulary.

[0008]   Unfortunately, heavy reliance on spelling models and alternative entry processes is simply impractical for a general-purpose typing recognizer. Typing, after all, is the fallback entry process for many handwriting and speech recognition systems, and the only fallback conceivable for typing is a slower, clumsier typing mode. Likewise, personal computer users have to type into a wide variety of applications requiring strange character strings like passwords, filenames, abbreviated commands, and programming variable names. To avoid annoying the user with frequent corrections or dictionary additions, spelling model influence must be weak enough that strings missing from it will always be accepted when typed at moderate speed with reasonable care. Thus a general-purpose typing recognizer should only rely on spelling models as a last resort, when all possible measurements of the actual typing are ambiguous.

Brief Summary of the Invention

[0009]   Since a typing recognizer cannot depend too much on spelling models, there still exists a need in the touchscreen and touch keyboard art for spelling-independent methods to improve recognition accuracy. The main aspect of the present invention is to search for the geometric pattern of keys that best matches the geometric pattern of a touch sequence, rather than just searching for the key closest to each touch. This method improves recognition accuracy without any assumptions about the character content being typed.

[0010]   According to this aspect of the invention, touch or finger stroke coordinates reported by a sensing device and key coordinates from a key layout feed into a typing recognizer module. The typing recognizer then hypothesizes plausible sequences of keys by extending existing sequences with keys that are within the immediate neighborhood of the newest finger touch. It can also hypothesize home row key locations for touches caused by hands resting on or near the home row keys. For each hypothesized sequence, the typing recognizer computes separation vectors between the layout position of successive keys in the sequence. The typing recognizer also computes separation vectors between successive touch positions in the touch sequence. Each key sequence is evaluated according to a pattern match metric that includes not only the distance between each finger touch and the corresponding key but also how closely the separation vectors between successive touches match the separation vectors between successive keys. The hypothesized sequence with the best cumulative match metric is transmitted to the host computer, possibly replacing an older, higher cost partial sequence that was transmitted previously.

[0011]   It is therefore an objective of this invention to provide typing recognition methods that overcome the shortcomings of the related touchscreen and touch keyboard art.

[0012]   A primary objective of the present invention is to recognize typing accurately even when lack of tactile key position feedback leads to significant hand and finger drift.

[0013]   Yet another objective of this invention is to improve typing recognition

accuracy without excessive dependence on spelling models.

[0014]   A further objective of this invention is to disambiguate typing as much as possible with measurements of its geometric pattern before falling back on a spelling model to resolve any remaining recognition ambiguities.

[0015]   A secondary objective of this invention is to beneficially incorporate key/hand alignment measurements from resting hands into recognition decisions without explicitly shifting the key layout into alignment with the resting hands.

Brief Description of the Drawings

[0016]   Figure 1 is a block level diagram of the preferred surface typing detection and recognition system for the present invention.

[0017]   Figure 2 contains illustrations of a sample touch sequence on the left half of a standard QWERTY key layout (Figure 2 A), the touch separation vectors for the sample touch sequence (Figure 2 B), and the key separation vectors for several hypothesized key sequences that might correspond to the key sequence intended by the touch typist (Figure 2 C-J).

[0018]   Figure 3 illustrates the contents of the touch data structure used to store measured touch parameters, a decoding stack, and key finally output for a touch.

[0019]   Figure 4 illustrates the contents of the hypothesis data structure that serves as nodes of the hypothesis trees for the present invention.

[0020]   Figure 5 is a flow chart illustrating the preferred embodiment of key hypothesis tree generation according to the present invention.

[0021]   Figure 6 is a diagram illustrating a hypothesis tree that could be generated by the process of Figure 5 during recognition of the sample touch sequence in Figure 2 .

[0022]   Figure 7 is a flow chart illustrating the steps for computing the geometry match metric of each key hypothesis.

[0023]   Figure 8 is a flow chart illustrating the process that outputs the best new key hypothesis to the host computer, erasing as necessary previously output keys that differ from past keys in the current best sequence.

Detailed Description of the Invention

[0024]   In the preferred embodiment, the typing recognition methods of this invention are utilized within a multi-touch system like that shown in Figure 1 . The sensor scanning hardware 6 detects touches by fingers 2 on the surface 4. The proximity image formation 8 and contact tracking 10 modules determine the touch timing and surface coordinates and report these to the typing recognizer 12. The typing recognizer decides which keys the user intended to press and tells the host communications interface 16 to send those keys to the host computer 18. The system may also include a chord motion recognizer module 14 that interprets lateral sliding of multiple fingers as pointing or gesture input and effectively disables the typing recognizer for such touches. The synchronization detector 13 searches for simultaneous presses or releases of multiple fingers, thereby aiding in detection of chord slides, chord taps, and resting hands. All modules besides the typing recognizer are fully described in related U.S. Pat. App. Serial No. 09/236,513 by Westerman and Elias. That application is incorporated herein by reference in its entirety. The present invention constitutes improvements to the rudimentary 'nearest key' typing recognizer described in that application.

[0025]   Those skilled in the art will recognize that the typing recognizer disclosed herein could be utilized with any sensing device that accurately reports the lateral position of fingertips as they near the end of their stroke, whether or not the fingers actually touch a surface or depress physical keys. Examples of such alternative finger position sensing systems include micro radar, data gloves, and pressure-sensitive surface materials. The term touch location will be used hereafter for the lateral position or x and y coordinates detected for fingertips within a plane roughly normal to the fingertips at the end of their stroke, even for sensing devices that require no physical contact with a surface at the end of the stroke. Likewise, the typing recognition software need not reside within a microprocessor packaged with the sensing device. It could just as easily execute

within the host computer system, or the host computer system and sensing device might be combined such that the same microprocessor executes finger tracking, typing recognition, and user application software.

[0026]   Related art 'nearest key' typing recognizers typically assume that touch location errors are independent from keystroke to keystroke. But for typing devices that don't provide strong tactile feedback of key position, the hand sometimes drifts slightly out of alignment with the key layout. This causes the absolute location errors for most touches to be biased in the drift direction and statistically dependent. However, if the typist still reaches the proper amount (a whole number of key spacings) relative to recent touches, the lateral separations between finger touches will closely match the separations between the keys the typist intended to strike, regardless of the overall hand drift.

[0027]   A related type of bias occurs when individual fingers drift relative to the rest of the hand. This causes the absolute location errors to be biased the same way for all keys typed by the drifting finger(s). However, keys typed by adjacent fingers may not share this bias.

[0028]

An important discovery of the present invention is that when trying to recognize a sequence of touches located ambiguously between keys, searching for key sequences whose relative geometric pattern matches the touch pattern greatly narrows the list of plausible key sequences. This is illustrated intuitively in Figure 2 . Figure 2 A shows a series of four touches as triangles *t0, t1, t2, t3* , on the left half of a QWERTY key layout 29. The distance between a given key and touch, herein referred to as the zero-order key/touch alignment error, is apparent by inspection. The radii of the dotted circles 30 indicate the distance from a touch to the *nearest* key. Touch *t0* is roughly equidistant from keys 'D' and 'F', as indicated by *t0* 's circle passing through both key symbols, and *t0* is not far from 'C' or 'V' either. A 'nearest key' recognizer would associate *t0* with 'D', but with little confidence. If *t0* was just a bit farther right, 'F' would become the nearest choice. A nearest key recognizer also faces a tossup between 'E' and 'R' for t3, and cannot be terribly confident of recognizing *t2* as 'R'. Touch *t1* is the only touch close enough

to a single key ('A') to be confidently interpreted as that key.

[0029]   Figure 2 B illustrates the vectors separating successive touches. Solid lines 32 are 'first-order' vectors from $t0$ to $t1$, $t1$ to $t2$, and $t2$ to $t3$. Dashed lines 34 are 'second-order' vectors from $t0$ to $t2$ and $t1$ to $t3$. The dotted line 36 is the 'third-order' vector from $t0$ to $t3$. Figures 2 'C'-'H' show corresponding key separation vectors for possible matching key sequences. In all cases but Figure 2H and Figure 2J , at least one of the key separation vectors clearly differs from a corresponding touch separation vector. For the 'CARE' hypothesis in Figure 2C , the third-order 'C'-'E' vector is significantly longer than the corresponding $t0-t3$ vector. For the 'FARE' hypothesis in Figure 2D , the second-order 'F'-'R' and third order 'F'-'E' vectors have clearly different angles than the corresponding $t0-t2$ and $t0-t3$ vectors. For the 'CARR' and 'DARR' hypotheses in Figures 2E and 2G , the first order 'R'-'R' vector will have length 0, quite different than the first order $t2-t3$ vector's length of one full key-spacing. For the 'FATE' hypothesis of Figure 2F , the 'T'-'E' vector is now a full key-spacing longer than the $t2-t3$ vector. Even though all the hypotheses shown are nearly indistinguishable in terms of the zero-order alignment error between each touch and corresponding key, an improved typing recognizer that compares the touch separation and key separation vectors can quickly eliminate all but hypotheses 'DARE' and 'FSRT' in Figures 2H and 2J . The final decision can be made based upon 'DARE's smaller zero-order, absolute error between $t1$ and 'A' than between $t1$ and 'S'. In even more ambiguous cases, a language model can help choose English words (like 'DARE' instead of 'FSRT') from the list of remaining hypotheses.

[0030]   Since typists expect the symbol of each touched key to appear on the host computer screen immediately after each corresponding finger stroke, a typing recognizer cannot wait for an entire touch sequence to complete before choosing the best key sequence. In a preferred embodiment of this invention, the recognizer module decodes the touch sequence incrementally, extending key hypothesis sequences by one key each time a new touch is detected. This process will form a hypothesis tree whose nodes are individual key hypotheses. It is important to note that related art 'nearest key' recognizers need not construct a hypothesis tree since

they assume that finger placement errors from each keystroke are statistically independent.

[0031]   Figure 3 lists the basic parameters the recognizer needs to store in each touch data structure 79. A ring or chain of such data structures ordered by touchdown time represents a touch sequence. Each touch data structure 79 must contain the touch's x and y surface coordinates 70 as reported by the touch sensors. These should estimate the center of the touch, which for proximity or pressure sensors is typically computed as the centroid of fingertip flesh contacting the surface. To help lookup the home row key of each touch from a resting hand, each touch data structure should have a copy of the hand and finger identity 71 estimated for the touch by the contact tracking and identification module 10. To keep track of the recency of past touches, the touch data should also include the finger touchdown time or press time 72. For compressible surfaces, this should correspond to the time the finger stroke bottomed out. The touch release time 73 should be set to either the the time of finger liftoff from the surface or the current system time if the finger is still touching. To aid in decoding the most likely hypothesis sequence, all hypotheses caused by a touch will be inserted into a stack 76 and sorted so that the hypothesis with the best cumulative metric 98 rises to the top of the stack. Finally, to support undoing preliminary key outputs, the touch structure should maintain a reference 77 to the hypothesis whose key gets output in response to the touch. This reference will be *null* until a key is chosen to be output through the host communications interface 16.

[0032]   Figure 4 shows that to establish the tree structure, each hypothesis data structure 85 needs a reference 86 to its parent hypothesis from the previous touch. For the very first touch, this reference will be *null*, representing the root of the hypothesis tree. Having a reference to the data structure 88 of the touch causing the hypothesis is also convenient. The key center coordinates 92, and key code, symbol or command to be output 94, are retrieved from the key layout according to which key the hypothesis represents. Once computed, a measure of the match between the touch pattern and key pattern represented by the key hypothesis and its parent sequence will be stored as the geometry match metric 96. Though the

embodiment presented herein formulates this geometry match metric as a cost to be minimized, it can just as easily be formulated as a probability to be maximized and remain well within the scope of this invention. It will be added in step 222 of Figure 7 to the parent 86 hypothesis' cumulative match metric to obtain a new, extended cumulative match metric 98 for the sequence. In embodiments that include a spelling model, each hypothesis data structure 85 will also need to hold a spelling match metric 97. The spelling match metric may also be formulated as either a bad spelling cost to be minimized or a character transition probability to be maximized.

[0033]   Figure 5 is a flowchart illustrating the preferred embodiment of the hypothesis tree extension, evaluation, and decoding processes. Step 100 shows that the typing recognizer starts up with the touch count n set to 0 and the hypothesis tree empty. Decision diamond 102 waits for a new touch to be detected by the sensors and recorded as $T[n]$, the newest touch data structure 79 of the chain. We will use the pseudo-code notation $T[n].x$ and $T[n].y$ for the touch coordinates 70. Step 104 resets the parent hypothesis index p to 0. Step 106 retrieves a parent hypothesis $h_p[n-1]$ data structure 85 associated with the previous touch $T[n-1]$. In the case that n equals 0, step 106 simply sets the parent hypothesis to $null$, representing the root of the empty tree. Step 108 resets the new hypothesis counter j to 0. Step 110 picks a key from the key layout, an array of key coordinates and symbols that describes the arrangement of keys across the surface. Decision diamond 112 tests whether the key center is within a maximum activation radius R_{act} of the new touch $T[n]$'s surface coordinates. If the key is too far away, it need not be evaluated further, and decision diamond 111 will pick another key from the layout 110 until all keys in the vicinity of the touch have been hypothesized. About one standard key-spacing (~2 cm or 3/4" inch) is sufficiently large for R_{act}, but R_{act} can be bumped up for oversize keys like $Space$, $Shift$, and $Enter$. Choosing R_{act} too large wastes computation by hypothesizing keys that are nowhere near the finger touch and that the typist clearly did not intend to hit. Choosing R_{act} too small limits the amount of hand drift that the typing recognizer can correct for.

[0034]   If a key is within the radius R_{act} of the new touch, step 114 creates a new

hypothesis $h_j[n]$ (using data structure 85) descended from the current parent $h_p[n-1]$. The new hypothesis' parent hypothesis reference 86 is set accordingly. Block 116 evaluates how well the new key hypothesis $h_j[n]$ and its parent sequence matches the touch sequence $T[0]...T[n]$. Figure 7 will describe this critical block in more detail. Step 118 inserts the new hypothesis $h_j[n]$ into $T[n]$'s stack 76, which is sorted such that hypotheses with the best cumulative match metric (either lowest sum of costs or highest product of probabilities) rise to the top.

[0035]   Once hypotheses descended from parent $h_p[n-1]$ have been generated for all keys near the touch $T[n]$, decision diamond 120 decides whether the previous touch $T[n-1]$'s stack 76 contains additional parent hypotheses that need to be extended. If so, the parent hypothesis index p is incremented in step 122, and steps 106-122 repeat for the next parent. Once all parent hypotheses have been extended, block 124 actually outputs the best hypothesis sequence as described further in Figure 8 . Step 126 prunes from the tree those hypotheses whose cumulative match metric is already so poor that they are very unlikely to spawn best hypotheses in the future. This prevents exponential growth of the hypothesis tree by discarding clearly bad hypotheses but preserving competitive hypotheses that might become parents of the best hypothesis for a future touch. The most efficient pruning method is to start at the bottom of $T[n]$'s stack 76 and discard all hypotheses whose cumulative metric is not within a future cost margin of the top (best) hypothesis's cumulative match metric. When all of a parent's child hypotheses have been discarded the parent is discarded as well. The pruning step 126 completes all processing of touch $T[n]$, leaving step 128 to increment the touch index n so decision diamond 102 can resume waiting for the next touch.

[0036]   Working together, steps 118, 124, and 126 constitute a stack decoder. They sort all of the new hypotheses for the current touch $T[n]$ according to their cumulative match metric, choose the lowest cost sequence that winds up at the top of the stack as the best hypothesis sequence to output, and prune the implausible sequences at the bottom of the stack whose costs are much greater than the current best sequence. The stack decoder is a well-known method in the speech

recognition, handwriting recognition, and digital communications arts for finding the optimal path through a hypothesis tree. For example, see F. Jelinek, Statistical Methods for Speech Recognition (published by The MIT Press, pages 93-110, 1997). Those skilled in the art will recognize that a basic Viterbi decoder would only be appropriate in place of the stack decoder if the touch geometry metric only included first order separation vectors. Including higher order separation vectors as is necessary to get a wholesome hand drift estimate makes the touch cost dependent on more than the previous touch and thus violates the first-order Markov condition for basic Viterbi decoders.

[0037]   Figure 6 shows an example of a hypothesis tree that the typing recognition process in Figure 5 might generate while decoding the touch sequence described in Figure 2 . The tree starts empty while waiting for the first touch, consisting only of the *null* root 150. When touch *t0* 152 is detected, the typing recognizer will sprout hypotheses 154 for the keys 'D', 'F', and 'C' neighboring *t0* . Because the sequence so far contains only one touch, the match metric for these first keys will only include the zero-order, key/touch alignment error distance. In this case, the typing recognizer would be ready to output the 'D' key since, referring to Figure 2A , 'D' is closest to *t0* . When touch t1 arrives 156, each hypothesis for t0 branches into hypotheses 158 for the keys nearest *t1* , namely 'A' and 'S'. The match metric for these *t1* hypotheses can include both the zero-order key/touch alignment error and first-order separation vectors between *t1* and *t0* . With a second touch, the typing recognizer is ready to start picking the best hypothesis *sequence* . To do so, for each *t1* hypothesis it must compute a cumulative cost that also includes the cost of the parent *t0* hypothesis. The *t1* hypothesis with lowest cumulative cost will be selected, in this case 'DA'. Since 'D' was just output, only 'A' need be sent to the host.

[0038]   In case the previous touch's output had been some key other than 'D', say 'F', the preliminary 'F' output would need to be undone and replaced with 'D' by sending a *BackSpace* or *Erase* key followed by 'DA' to the host. The hypothesis tree extensions and output of best sequence would continue similarly for the t2 and t3 touches, except that the match metrics for these touches would include second

and third-order separation vectors, respectively. Pruning of hypothesis chains 160 that accumulate relatively high total costs prevents the tree from growing exponentially as more touches occur.

[0039]   The flowchart in Figure 7 illustrates how the preferred embodiment of the typing recognizer evaluates the quality of the match between a hypothesized key sequence and the corresponding touch sequence. This expanded flowchart corresponds to step 116 of Figure 5 . For the convenience of those skilled in the art, the evaluation process is also shown belo as pseudocode:

```
Copy hⱼ[n] and its parent hypothesis sequence into hseq[n]...hseq[0]
for (m=0; m < 10 && n-m >= 0; m++) {
        if (m == 0) { //zero-order key/touch alignment error
            hseq[n].geomcost = d0(T[n].x - hseq[n].x,T[n].y - hseq[n].y)
            continue;
        } else if (T[n].hand_identity != T[n-m].hand_identity)
            continue;
        else if(T[n-m] not keystroke or resting finger)
            break;
        τ[m].x = T[n].x - T[n-m].x  //touch separation vectors
        τ[m].y = T[n].y - T[n-m].y
        λ[m].x = hseq[n].x - hseq[n-m].x  //key separation vectors
        λ[m].y = hseq[n].y - hseq[n-m].y
        wₜ[m] = fₜ(T[n].tpress-T[n-m].trelease)
        wₐ[m] = fₐ(τ[m].x,τ[m].y)
        hseq[n].geomcost +=wₜ[m]*wₐ[m]*
                            dM(τ[m].x-λ[m].x,τ[m].y-λ[m].y)
}
hseq[n].cumulcost = hseq[n].geomcost + hseq[n-1].cumulcost
```

[0040]   For notational and computational convenience, step 200 copies the particular key hypothesis sequence to be evaluated into the array *hseq[]* , starting at *hj[n]* , the new leaf of the hypothesis tree, traversing back through its parent hypothesis references, and stopping at the tree root. Step 202 computes the zero-order, key/touch misalignment error and stores it as the hypothesis' geometry match metric 96, *hseq[n].geomcost* . The distance metric *d0* determines how the *hseq*

[n].geomcost scales with misalignment in the *x* and *y* dimensions. Those skilled in the art will realize that any of a Manhattan metric, Euclidean distance, squared Euclidean distance metric or other metrics would be suitable here. Related art 'nearest key' typing recognizers essentially stop with this zero-order alignment error as the final geometry metric, but the current invention includes higher order separation vector mismatches in the geometry metric via the following steps.

[0041]   Step 204 initializes the order index *m* to 1. Since each hand's drift is presumed to be independent of the other's drift, only separation vectors for touches and keys typed within the same hand should be considered. Decision diamond 206 tests whether the *m* th previous hypothesized key *hseq[n-m]* is normally typed by the same hand as the currently hypothesized key *hseq[n]* . If not, *hseq[n-m]* presumably contains no information about the drift of the current hand, so the evaluation process skips *m* th-order separation vector computations and advances to step 218.

[0042]   If both touches come from the same hand, decision diamond 207 decides whether the *m* th previous was actually typing related and thus a possible predictor of hand drift. Decision diamond 207 is particularly important for multi-touch systems that support non-typing synchronous touches such as chord taps, lateral chord slides, and hand resting. For instance, finger location at the beginning or end of pointing motions has nothing to do with subsequent typing drift, so decision diamond 207 should break the loop and skip to the final cost accumulation step 222 when it encounters a touch involved in pointing or any other sliding gesture. However, when typing on a surface, resting a hand (all fingers simultaneously) on home row in between words is quite convenient. Any slight misalignments between the home row keys and finger locations within the resting chord are a good predictor of hand/key misalignment during subsequent typing. Such resting finger locations can be incorporated into separation vector evaluation by having the synchronization detector 13 insert a chain of five special resting finger hypotheses into the hypothesis tree for any five nearly simultaneous touches deemed to be part of a hand resting on or near its home row keys. Each resting finger hypothesis is given key coordinates 92 from the home row key that

its finger normally rests on. The hypothesis can look up its finger and hand identity 71 through its causing touch reference 88, and the identities can then index into a table of home row key center coordinates. Resting finger hypotheses are given a null key code 94 so that they produce no output signals to the host computer. For the purpose of key and touch separation vector matching, however, decision diamond 207 and steps 208-216 of <u>Figure 7</u> treat them as typing-related hypotheses. This subtle incorporation of resting hand alignment is an alternative to the key layout morphing method described by Westerman and Elias in U.S. Pat. App. Serial No. 09/236,513. The morphing method snaps the home row keys to the precise resting finger locations and shifts the rest of the key layout accordingly, thus removing any misalignment between the resting hand and the key layout, but is only practical for touch surfaces integrated onto a video display that indicates key location shifts to the user.

[0043]   For typing-related touches from the same hand, step 208 creates the m th-order touch separation vector $i_s[m]$ by subtracting the spatial and temporal coordinates of the m th previous touch $T[n-m]$ from the current touch $T[n]$. Likewise, step 210 creates the m th-order key separation vector $i_s[m]$ by subtracting the layout coordinates of $hseq[n-m]$'s key from the currently hypothesized key $hseq[n]$.

[0044]   Step 212 computes the temporal confidence weighting $w_t[m]$ that should decrease monotonically toward 0 with the time elapsed between the press 72 of the current touch, $T[n].tpress$ and release 73 of the m th previous touch, $T[n-m].trelease$. The release time is used in case the preceding touch was caused by a hand that began resting near home row many seconds ago but lifted off quite recently. This temporal confidence weighting is meant to reflect the fact that old touches are poorer predictors of the current hand drift than newer touches. Those skilled in the art will realize that the exact downward slope for this weighting function can be empirically optimized by computing old and new touch drift correlations from actual typing samples. For instance, if the typing samples showed that the hand/layout alignment error remained fairly consistent over ten second periods, then the weighting function should be designed to stay well above

0 for touches less than ten seconds old.

[0045] Step 214 computes a touch adjacency weighting $w_a[m]$ that should decrease monotonically toward 0 as the separation between the current and m th previous touch increases. The touch adjacency weighting is meant to reflect the fact that the separation between touches by the same finger or an adjacent finger, especially if the fingers have not reached far between the touches, is a better predictor of finger drift and overall hand drift than separation vectors for touches by non-adjacent fingers. Thus the second-order separation vector between $t2$ and $t0$ in Figure 2B should be weighted more heavily than the long, first-order separation vector between $t2$ and $t1$. The adjacency weighting should be strongest when the m th previous touch occurred at the same surface location as the current touch, as this is a very strong indication both touches were intended to produce the same key. In this situation, the m th order key separation vector $\hat{b}[m]$ of the matching key sequence is expected to have zero length, and any hypothesized key sequences with a non-zero m th order vector length should be punished with a strongly weighted cost.

[0046] Step 216 adds to the geometry metric a cost for any mismatch between the m th-order touch separation vector $\hat{l}_n[m]$ and the m th-order key separation vector $\hat{b}[m]$. This incremental cost should generally increase with the magnitude of the difference between the two vectors. In the preferred embodiment, the square of the magnitude of the vector difference is weighted by the temporal confidence $w_t[m]$ and adjacency confidence $w_a[m]$ to obtain the m th-order cost increment. The squared Euclidean metric is preferred for dM because it favors sequences with uniformly small vector differences.

[0047]

Step 218 increments the order index m so that decision diamond 220 can decide whether to continue evaluating higher order separation vectors. Ideally, the evaluation process would continue with previous touches all the way back to the tree root, where m reaches n, but in practice it is usually sufficient to include separation vectors from the ten or so most recent typing-related touches. Once decision diamond 220 decides m has reached its useful limit, flow falls through to

the final step 222. Step 222 sets the sequence cumulative match metric h_j [n].cumulcost to the sum of the new touch cost hseq[n].geomcost and the parent's cumulative metric hseq[n-1].cumulcost.

[0048]   It is also instructive to examine an alternative embodiment of geometry match metric evaluation that, mathematically, is the exact equivalent of and produces the same result as the process in Figure 7 . However, a different factoring of the computations lends this alternative embodiment a differently intuitive interpretation. For the convenience of those o ordinary skill in the art, this alternative embodiment is shown below as pseudocode:

```
Copy hj[n] and its parent hypothesis sequence into hseq[n]...hseq[0]
Allocate key/touch error array e[] for different orders
for (m=0; m < 10 && n-m > 0; m++) {
        e[m].x = T[n-m].x - hseq[n-m].x  //alignment errors
        e[m].y = T[n-m].y - hseq[n-m].y
        if (m == 0) { //zero-order key/touch alignment error
            hseq[n].geomcost = d0(e[0].x,e[0].y)
            continue;
        } else if (T[n].hand_identity != T[n-m].hand_identity)
            continue;
        else if(T[n-m] not keystroke or resting finger)
            break;
        wt[m] = ft(T[n].tpress-T[n-m].trelease)
        τ[m].x = T[n].x - T[n-m].x   //touch separation vectors
        τ[m].y = T[n].y - T[n-m].y
        wa[m] = fa(τ[m].x,τ[m].y)
        hseq[n].geomcost +=wt[m] * wa[m] *
                            dM(e[0].x-e[m].x,e[0].y-e[m].y)
}
hseq[n].cumulcost = hseq[n].geomcost + hseq[n-1].cumulcost
```

Both embodiments compute the zero-order alignment error component the same, but this alternative embodiment restates the comparison between the m th-order key and touch separation vectors as a comparison between the new touch T[n]'s key/touch alignment erro vector, e[0], and the m th previous touch T[n-m]'s key/touch alignment error vector, e[m] . This suggests that the stack decoder in either embodiment will tend to pick as the best

sequence a key hypothesis sequence whose individual key/touch alignment error vectors ar small yet consistent with one another. Clearly this alternative, equivalent embodiment falls well within the scope of this invention.

[0049]   The output module in Figure 8 is responsible for transmitting the key code, command or symbol 94 from the best hypothesis *hbest[n]* to the host application. This job is complicated by the fact that any keys sent for previous touches may not have come from *hbest[n]*'s parent sequence. This happens when, based on additional cost evaluations from a new touch, a stack decoder decides a totally different sequence is optimal than was considered optimal from previous touch information alone. This occurrence presents the human interface designer with a tough question—leave the old character sequence or partial word on the screen, even though the new key is likely to be from a different word, or erase characters that have already been displayed to the typist and replace them with the better sequence. This question is important because in rare instances the old characters may actually be what the user intended to type, in which case replacing them with the new, supposedly more optimal sequence will annoy and surprise the typist.

[0050]   The preferred embodiment of the output module adopts a compromise. It will only replace characters within the current word (*i.e.* it will not go back past any space characters and change any completed words), and it will only replace these characters if they have only been typed within the last couple seconds, before the typist has had a chance to notice and correct the probably erroneous old characters himself. The output module starts with the current best hypothesis 350 *hbest[n]* from the stack decoder. Step 352 sets the previous output index *m* to 1. Decision diamond 354 checks whether the hypothesis 77 whose key was output for touch *T[n−m]* was *hbest[n]*'s parent hypothesis *hbest[n−m]* . If not, decision diamond 356 checks whether the old key was a word-breaking space or was output more than a few seconds ago. If not, step 358 sends an *Erase* or *Backspace* key to the host to undo the old character, and step 360 increments *m* to continue checking for a parent hypothesis that both the best sequence and previously sent sequence share. Once that parent is found or the search is aborted at a word boundary, step 362 begins sending the replacement key codes 94 from the *hbest[]*

sequence, looping through step 363 to increment *m* until decision diamond finds that *m* has reached 0, and *hseq[n]*'s key code 94 has been transmitted.

[0051]   Now that the preferred embodiment of the typing recognizer has been described, it is instructive to consider additional consequences of its design. One important consequence is that the key activated may not always be the key nearest the fingertip. Generating a neighboring key when the finger actually lands right on top of another key would be startling to the user. However, if the adjacency weightings are kept sufficiently low, the separation vectors cannot override a zero-order, key/touch position error near zero. Proper tuning of the adjacency weighting function ensures that separation vectors can only be decisive when the finger lies in a zone between keys, at least 2-4 mm (1/8"-1/4") from the center of any key.

[0052]   To further improve recognition accuracy when typing plain English or another predictable language, alternative embodiments of the typing recognizer may incorporate a spelling model. Such integration of spelling models into character recognizers is clearly taught in the handwriting recognition art (see, for example, the post-processing with Markov model and Dictionary in U.S. Patent No. 5,812,698 to Platt *et al.* and the use of trigrams in U.S. Patent No. 6,137,908), and will only be summarized here briefly. Basicly, the spelling model computes for each hypothesis a character transition cost that indicates whether the hypothesized key/character is building a dictionary word out of its parent hypothesis sequence. Costs will be high for character transitions that cannot be found in the dictionary. Command or editing keys can be given a neutral or zero spelling cost. Step 222 of Figure 7 can then be modified to include the character transition cost weighted with the geometry cost in the cumulative cost total. Character transition costs need only be determining of the best sequence when different hypothesized key sequences have equally high touch geometry costs.

[0053]

    The case of a finger repetitively striking the same location halfway between keys is a good example of the advantages of considering touch sequence geometry in addition to zero-order alignment error, especially for typing recognizers that

include a spelling model. Typists find it disconcerting if they strike the same location repeatedly yet the decoder outputs different neighboring characters. This can happen, say, if the user intended to type 'DDD' but the three consecutive finger strikes occur roughly between the 'S', 'E', and 'W' and 'D' keys. For a 'nearest key' recognizer with spelling model, the zero-order alignment errors for all four keys would be roughly equal, leaving the character transition costs to dominate and encourage the stack decoder to output common spelling sequences like 'WES', 'SEW', and 'DES.' But for a typing recognizer improved with touch geometry matching, only the key sequences 'SSS', 'EEE', 'DDD' and 'WWW' have small key separation vectors matching the small touch separations, so these sequences' relatively low geometry match costs would override the spelling model, causing one of them to be output. Even though the 'SSS' or 'EEE' sequences may not be what the typist intended, they are less disconcerting than a mixed output sequence like 'SEW' when the typist knows her finger was not hopping between keys. Thus separation vector matching can overcome misleading character transition costs to ensure the typist sees a consistent, homogeneous output sequence when a finger strikes approximately the same location repeatedly.

[0054]   Though embodiments and applications of this invention have been shown and described, it will be apparent to those skilled in the art that numerous further embodiments and modifications than mentioned above are possible without departing from the inventive concepts disclosed herein. The invention, therefore, is not to be restricted except in the true spirit and scope of the appended claims.